2

AD-A210 028

(When Data Entered)

IENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: CAP Industry Ltd., CAPTACS-E286, V2.1, MICROVAX (Host) to INTEL 80286 (Target), 880528N1.09065	5. TYPE OF REPORT & PERIOD COVERED 27 May 1988 to 27 May 1989 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Wright-Patterson AFB Dayton, OH, USA	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Wright-Patterson AFB Dayton, OH, USA	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081	12. REPORT DATE 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (If different from Controlling Office) Wright-Patterson AFB Dayton, OH, USA	15. SECURITY CLASS (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

DTIC ELECTE JUN 30 1989 S E D

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20. If different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

CAP Industry Ltd., CAPTACS-E286, V2.1, MICROVAX under MICROVMS, V4.6 (Host) to INTEL 80286 bare computer (Target), ACVC 1.09.

89 6 29 170

DD FORM 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

Ada* Compiler Validation Summary Report:

Compiler Name: CAPTACS-E286, V2.1

Certificate Number: #880528N1.09065

Host:	Target:
MICROVAX under MICROVMS, V4.6	INTEL 80286 bare computer

Testing Completed 27 May 1988 Using ACVC 1.9

This report has been reviewed and is approved.

J.Pink

The National Computing Centre Ltd
Jane Pink
Oxford Road
Manchester, M1 7ED
United Kingdom

DTIC COPY INSPECTED 6

Accession For	
NTIS GRA&I	☒
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

[illegible]

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

[illegible]

Ada Joint Program Office
Dr. John Solomond
Director
Washington D.C. 20301

* Ada is a registered trademark of the United States Government *(Ada Joint Program Office).

Ada* Compiler Validation Summary Report:

Compiler Name: CAPTACS-E286, V2.1

Certificate Number: #880528N1.09065

Host:	Target:
MICROVAX under MICROVMS, V4.6	INTEL 80286 bare computer

Testing Completed 27 May 1988 Using ACVC 1.9

This report has been reviewed and is approved.

J Pink

The National Computing Centre Ltd
Jane Pink
Oxford Road
Manchester, M1 7ED
United Kingdom

[illegible]

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA 22311

Ada Joint Program Office
Dr. John Solomond
Director
Washington D.C. 20301

*** Ada is a registered trademark of the United States Government *(Ada Joint Program Office).**

AVF Control Number: AVF-VSR-90502/18

Ada COMPILER VALIDATION SUMMARY REPORT:

Certificate Number: #880528N1.09065
CAP Industry Ltd
CAPTACS-E286, V2.1
HOST: MICROVAX
TARGET:INTEL 80286

Completion of On-site Testing:
27 May 1988

Prepared By:
The National Computing Centre Limited
Oxford Road
Manchester M1 7ED
United Kingdom

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C. 20301-3081

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability, (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada Compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from the characteristics of particular operating systems, hardware, or implementation strategies. All the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behaviour that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:-

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard

- To attempt to identify any language constructs not supported by the compiler but required by the Ada Standard

- To determine that the implementation-dependent behaviour is allowed by the Ada Standard.

Testing of this compiler was conducted by NCC under the direction of the AVF according to procedures established by the Ada Joint Program Office and administered by the Ada Validation Organization (AVO). On-site testing was completed 27 May 1988 at Orion Court, Kenavon Drive, Reading.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented.

Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:-

The National Computing Centre Ltd
Oxford Road
Manchester M1 7ED
United Kingdom

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:-

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.

2. Ada Compiler Validation Procedures and Guidelines, Ada Joint Program Office, 1 January 1987.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., December 1986.

4. Ada Compiler Validation Capability User's Guide, December 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. The set of Ada programs that tests the conformity of an Ada compiler to the Ada programming language.
Ada Commentary	An Ada Commentary contains all information relevant to the point addressed by a comment on the Ada Standard. These comments are given a unique identification number having the form AI-ddddd.
Ada Standard	ANSI/MIL-STD-1815A, February 1983 and ISO 8652-1987.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. The AVF is responsible for conducting compiler validations according to procedures contained in the Ada Validation Procedures and Guidelines.
AVO	The Ada Validation Organization. The AVO has oversight authority over all AVF practices for the purpose of maintaining a uniform process for validation of Ada compilers. The AVO provides administrative and technical support for Ada validations to ensure consistent practices.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	An ACVC test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.
Inapplicable test	An ACVC test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	An ACVC test for which a compiler generates the expected result.

Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or a combination of features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	An ACVC test found to be incorrect and not used to check conformity to the Ada Standard. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce compilation or link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. There are no explicit program components in a Class A test to check semantics. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers permitted in a compilation or the number of units in a library--a compiler may refuse a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles sucessfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support are self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of REPORT and CHECK_FILE is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation is provided in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. The applicability of a test to an implementation is considered each time the implementation is validated.
A test that is inapplicable for one validation is not necessarily inapplicable for a subsequent validation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of this validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: CAPTACS-E286, V2.1		
ACVC Version: 1.9		
Certificate Number:		#880528N1.09065
Host Computer:		
	Machine:	MICROVAX
	Operating System:	MICROVMS V4.6
	Memory Size:	10M Bytes
Target Computer:		
	Machine:	INTEL 80286
	Operating System:	bare computer
	Memory Size:	0.5M Bytes

Communications Network:	RS232C Serial Link

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behaviour of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. The tests demonstrate the following characteristics:

. Capacities.

The compiler correctly processes tests containing loop statements nested to 65 levels, block statements nested to 65 levels, and recursive procedures separately compiled as subunits nested to 17 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

. Universal integer calculations.

An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation processes 64 bit integer calculations. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

. Predefined types.

This implementation supports the additional predefined types LONG_INTEGER and LONG_FLOAT in the package STANDARD. (See tests B86001C and B86001D.)

. Based literals.

An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

. Expression evaluation.

Apparently some default initialization expressions for record components are evaluated before any value is checked to belong to a component's subtype. (See test C32117A.)

Assignments for subtypes are performed with the same precision as the base type. (See test C35712B.)

This implementation uses no extra bits for extra precision. This implementation uses all extra bits for extra range. (See test C35903A.)

No exception is raised when an integer literal operand in a comparison or membership test is outside the range of the base type. (See test C45232A.)

Apparently NUMERIC_ERROR is raised when a literal operand in a fixed-point comparison or membership test is outside the range of the base type. (See test C45252A.)

Apparently underflow is not gradual. (See tests C45524A..Z.)

. Rounding.

The method used for rounding to integer is apparently round to even. (See tests C46012A..Z.)

The method used for rounding to longest integer is apparently round to even. (See tests C46012A..Z.)

The method used for rounding to integer in static universal real expressions is apparently round away from zero. (See test C4A014A.)

. Array types

An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT for this implementation.

Declaration of an array type or subtype declaration with more than SYSTEM.MAX_INT components raises NUMERIC_ERROR. (See test C36003A.)

No exception is raised when 'LENGTH is applied to an array type with INTEGER'LAST + 2 components. (See test C36202A.)

No exception is raised when 'LENGTH is applied to an array type with SYSTEM.MAX_INT + 2 components. (See test C36202B.)

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises no exception. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises CONSTRAINT_ERROR when the length of a dimension is calculated and exceeds INTEGER'LAST. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternatively, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises no exception. (See test E52103Y.)

In assigning one-dimensional array types, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two- dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Discriminated types.

During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications during compilation. (See test E38104A.)

In assigning record types with discriminants, the expression appears to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

. Aggregates.

In the evaluation of a multi-dimensional aggregate, index subtype checks appear to be made as choices are evaluated. (See tests C43207A and C43207B.)

In the evaluation of an aggregate containing subaggregates, not all choices are evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

. Representation clauses.

An implementation might legitimately place restrictions on representation clauses used by some of the tests. If a representation clause is used by a test in a way that violates a restriction, then the implementation must reject it.

Enumeration representation clauses containing noncontiguous values for enumeration types other than character and boolean types are not supported. (See tests C35502I..J, C35502M..N, and A39005F.)

Enumeration representation clauses containing noncontiguous values for character types are not supported. (See tests C35507I..J, C35507M..N, and C55B16A.)

Enumeration representation clauses for boolean types containing representational values other than (FALSE => 0, TRUE => 1) are not supported. (See tests C35508I..J and C35508M..N.)

Length clauses with SIZE specifications for enumeration types are supported with the restriction imposed in Appendix F, point F.4. (See test A39005B.)

Length clauses with STORAGE_SIZE specifications for access types are supported. (See tests A39005C and C87B62B.)

Length clauses with STORAGE_SIZE specifications for task types are supported. (See tests A39005D and C87B62D.)

Length clauses with SMALL specifications are supported . (See tests A39005E and C87B62C.)

Record representation clauses are supported with the restriction imposed in Appendix F, point F.4. (See test A39005G.)

Length clauses with SIZE specifications for derived integer types are supported. (See test C87B62A.)

. Pragmas.

The pragma INLINE is not supported for procedures. The pragma INLINE is not supported for functions. (See tests LA3004A, LA3004B, EA3004C, EA3004D, CA3004E, and CA3004F.)

. Input/output.

The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, EE2201D, and EE2201E.)

The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101H, EE2401D, and EE2401G.)

The director, AJPO, has determined (AI-00332) that every call to OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behaviour for SEQUENTIAL_IO.

The director, AJPO, has determined (AI-00332) that every call to OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behaviour for DIRECT_IO.

The director, AJPO, has determined (AI-00332) that every call to OPEN and CREATE must raise USE_ERROR or NAME_ERROR if file input/output is not supported. This implementation exhibits this behaviour for TEXT_IO.

. Generics.

This compiler requires that a generic unit's body be compiled prior to instantiation.

Generic subprogram declarations and bodies can be compiled in separate compilations. (See tests CA1012A and CA2009F.)

Generic package declarations and bodies can be compiled in separate compilations. (See tests CA2009C, BC3204C, and BC3205D.)

Generic unit bodies and their subunits can be compiled in separate compilations. (See test CA3011A.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.9 of the ACVC comprises 3122 tests. When this compiler was tested, 27 tests had been withdrawn because of test errors. The AVF determined that 437 tests were inapplicable to this implementation. All inapplicable tests were processed during validation testing except for 174 executable tests that use file operations not supported by the implementation. Modifications to the code, processing, or grading for 31 tests were required to successfully demonstrate the test objective. (See section 3.6.)

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	103	1049	1440	17	11	44	2664
Inapplicable	6	3	413	0	7	2	431
Withdrawn	3	2	21	0	1	0	27
TOTAL	112	1054	1874	17	19	46	3122

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER													TOTAL
	2	3	4	5	6	7	8	9	10	11	12	13	14	
Passed	190	485	538	244	166	98	141	327	129	36	234	3	68	2659
Inapplicable	14	87	136	4	0	0	2	0	8	0	0	0	185	436
Withdrawn	2	14	3	0	0	1	2	0	0	0	2	1	2	27
TOTAL	206	586	677	248	166	99	145	327	137	36	236	4	255	3122

3.4 WITHDRAWN TESTS

The following 27 tests were withdrawn from ACVC Version 1.9 at the time of this validation:

B28003A	C35904A	C37215C	C41402A	CC1311B
	C35904B		C45332A	
E28005C	C35A03E	C37215E	C45614C	BC3105A
C34004A	C35A03R	C37215G	A74016C	AD1A01A
C35502P	C37213H	C37215H	C85018B	CE2401H
A35902C	C37213J	C38102C	C87B04B	CE3208A

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. The applicability of a test to an implementation is considered each time a validation is attempted. A test that is inapplicable for one validation attempt is not necessarily inapplicable for a subsequent attempt. For this validation attempt, 437 tests were inapplicable for the reasons indicated:

- A39005B is not applicable because the particular value chosen in this test is not supported by this compiler.

- A39005G is not applicable because the particular value chosen in this test is not supported by this compiler.

- C35502I..J (2 tests), C35502M..N (2 tests), C35507I..J (2 tests), C35507M..N (2 tests), C35508I..J (2 tests), C35508M..N (2 tests), A39005F, and C55B16A use enumeration representation clauses which are not supported by this compiler.

- C35702A uses SHORT_FLOAT which is not supported by this implementation.

- The following tests use SHORT_INTEGER, which is not supported by this compiler:

C45231B	C45304B	C45502B	C45503B	C45504B
C45504E	C45611B	C45613B	C45614B	C45631B
C45632B	B52004E	C55B07B	B55B09D	

- C45231D requires a macro substitution for any predefined numeric types other than INTEGER, SHORT_INTEGER, LONG_INTEGER, FLOAT, SHORT_FLOAT, and LONG_FLOAT. This compiler does not support any such types.

- C45531M, C45531N, C45532M, and C45532N use fine 48-bit fixed-point base types which are not supported by this compiler.

- C45531O, C45531P, C45532O, and C45532P use coarse 48-bit fixed base types which are not supported by this compiler.

- B86001D requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

- C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.

- CA2009C and CA2009F compile the bodies of generic units separately and folowing a compilation that contains instantiations of those units. This compiler requires that a generic units's body be compiled prior to instantiation, and so the unit containing the instatiations is made obsolete upon the compilation of the unit containing the bodies. The test fail to link.

- CA3004E, EA3004C, and LA3004A use the INLINE pragma for procedures, which is not supported by this compiler.

- CA3004F, EA3004D, and LA3004B use the INLINE pragma for functions, which is not supported by this compiler.

- AE2101C, EE2201D, and EE2201E use instantiations of package SEQUENTIAL_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.

- AE2101H, EE2401D, and EE2401G use instantiations of package DIRECT_IO with unconstrained array types and record types having discriminants without defaults. These instantiations are rejected by this compiler.

TEST INFORMATION

- The following 174 tests are inapplicable because sequential, and direct access files are not supported.

CE2102C	CE2102G..H(2)	CE2102K	CE2104A..D(4)
CE2105A..B(2)	CE2106A..B(2)	CE2107A..I(9)	CE2108A..D(4)
CE2109A..C(3)	CE2110A..C(3)	CE2111A..E(5)	CE2111G..H(2)
CE2115A..B(2)	CE2201A..C(3)	CE2201F..G(2)	CE2204A..B(2)
CE2208B	CE2210A	CE2401A..C(3)	CE2401E..F(2)
CE2404A	CE2405B	CE2406A	CE2407A
CE2408A	CE2409A	CE2410A	CE2411A
AE3101A	CE3102B	EE3102C	CE3103A
CE3104A	CE3107A	CE3108A..B(2)	CE3109A
CE3110A	CE3111A..E(5)	CE3112A..B(2)	CE3114A..B(2)
CE3115A	CE3203A	CE3301A..C(3)	CE3302A
CE3305A	CE3402A..D(4)	CE3403A..C(3)	CE3403E..F(2)
CE3404A..C(3)	CE3405A..D(4)	CE3406A..D(4)	CE3407A..C(3)
CE3408A..C(3)	CE3409A	CE3409C..F(4)	CE3410A
CE3410C..F(4)	CE3411A	CE3412A	CE3413A
CE3413C	CE3602A..D(4)	CE3603A	CE3604A
CE3605A..E(5)	CE3606A..B(2)	CE3704A..B(2)	CE3704D..F(3)
CE3704M..O(3)		CE3706D	CE3706F
CE3804A..E(5)	CE3804G	CE3804I	CE3804K
CE3804M	CE3805A..B(2)	CE3806A	CE3806D..E(2)
CE3905A..C(3)	CE3905L	CE3906A..C(3)	CE3906E..F(2)

Results of running a subset of these tests showed that the proper exceptions are raised for unsupported file operations.

- The following 201 tests require a floating-point accuracy that exceeds the maximum of 15 digits supported by this implementation:

C24113L..Y (14 tests)	C35705L..Y (14 tests)
C35706L..Y (14 tests)	C35707L..Y (14 tests)
C35708L..Y (14 tests)	C35802L..Z (15 tests)
C45241L..Y (14 tests)	C45321L..Y (14 tests)
C45421L..Y (14 tests)	C45521L..Z (15 tests)
C45524L..Z (15 tests)	C45621L..Z (15 tests)
C45641L..Y (14 tests)	C46012L..Z (15 tests)

3.6 TEST, PROCESSING, AND EVALUATION MODIFICATIONS

It is expected that some tests will require modifications of code, processing, or evaluation in order to compensate for legitimate implementation behaviour. Modifications are made by the AVF in cases where legitimate implementation behaviour prevents the successful completion of an (otherwise) applicable test. Examples of such modifications include: adding a length clause to alter the default size of a collection; splitting a Class B test into subtests so that all errors are detected; and confirming that messages produced by an executable test demonstrate conforming behaviour that wasn't anticipated by the test (such as raising one exception instead of another).

Modifications were required for 29 Class B tests, and 2 Class C tests.

. The following Class B tests were split because syntax errors at one point resulted in the compiler not detecting other errors in the test:

B27005A	B28001R	B28001V	B55A01A	B56001H
B71001B	B71001C	B71001D	B71001E	B71001F
B71001I	B71001K	B71001N	B71001O	B71001P
B71001Q	B71001R	B71001U	B71001W	B95077A
B97101A	B97101E	BA3006A	BA3006B	BA3007B
BA3008A	BA3008B	BA3013A	BC1202E	

. The following executable tests were split because the size of the code generated exceeded the segment limit on the target for a compilation unit:

C35A06N	CC1221A

. REPORT_BODY was modified by replacing all the occurances in the source code of TEXT_IO with REPORT_IO. REPORT_IO is a subset of TEXT_IO that directs standard input and output to the host computer via the RS232 connector.

. C45651A requires that the result of the expression in line 227 be in the range given in line 228; however, this range excludes some acceptable results. This implementation passes all other checks of this test, and the AVO ruled that this test is passed.

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.9 produced by the CAPTACS-E286 was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behaviour on all inapplicable tests.

3.7.2 Test Method

Testing of the CAPTACS-E286 using ACVC Version 1.9 was conducted on-site by a validation team from the AVF. The configuration consisted of a MICROVAX host operating under MICROVMS, V4.6, and a INTEL 80286 bare computer target. The host and target computers were linked via an RS232C Serial Link.

A magnetic tape containing all tests was taken on-site by the validation team for processing. Tests that make use of implementation-specific values were customized before being written to the magnetic tape. Tests requiring modifications during the pre- validation testing were not included in their modified form on the magnetic tape.

The contents of the magnetic tape were not loaded directly onto the host computer. The files were loaded from the magnetic tape to an RA60 disc via a VAX 11/750. The RA60 is dual ported to the MICROVAX. Files were copied from the RA60 to the fixed user disc on the MICROVAX.

After the test files were loaded to disk, the full set of tests was compiled and linked on the MICROVAX, and all executable tests were run on the INTEL 80286. Object files were linked on the host computer, and executable images were transferred to the target computer via an RS232 Serial Link. Results were printed from the VAX 11/750 after transferring them from the host computer via the RA60 as above.

The compiler was tested using command scripts provided by CAP Industry Limited and reviewed by the validation team. The compiler was tested using all default switch settings except for the following:

Switch	Effect
/LIST	Produce a compilation listing file.
/INPUT_LIST	Compile a list of source files.
/LIB_FILE	Specify library file name.

Tests were compiled, linked, and executed (as appropriate) using a single host computer and a single target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

Testing was conducted at Orion Court, Kenavon Drive, Reading and was completed on 27 May 1988.

APPENDIX A

DECLARATION OF CONFORMANCE

CAP Industry Limited has submitted the following Declaration of Conformance concerning the CAPTACS-E286.

DECLARATION OF CONFORMANCE

Compiler Implementor	CAP Industry Limited
Ada* Validation Facility	National Computing Centre Limited, UK
Ada Compiler Validation Capability (ACVC) Version	1.9

BASE CONFIGURATION

Base Compiler Name	CAPTACS-E286
Version	V2.1
Host Architecture	MICROVAX II
Host Operating System	MICROVMS
Version	4.6
Target Architecture	INTEL iAPX 80286 (protected mode)
Target Operating System	bare computer

I, the undersigned, representing CAP Industry Limited, have implemented no deliberate extensions to the Ada Language Standard ANSI-MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that CAP Industry Ltd is the owner of record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI-MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.

[illegible] Date: 25-5-88

Name of Person signing
Title : Project Manager

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

DECLARATION OF CONFORMANCE

Compiler Implementor : CAP Industry Limited

Ada* Validation Facility : National Computing Centre Limited, UK

Ada Compiler Validation Capability (ACVC) Version: 1.9

BASE CONFIGURATION

Base Compiler Name	CAPTACS-E286
Version	V2.1
Host Architecture	MICROVAX II
Host Operating System	MICROVMS
Version	4.6
Target Architecture	INTEL iAPX 80286 (protected mode)
Target Operating System	bare computer

DERIVED COMPILER REGISTRATION

Derived Compiler Name	CAPTACS-E286
Version	2.1
Host Architecture	MICROVAX II
Host operating System	MICROVMS
Version	4.6
Target Architecture	INTEL 80386
Target Operating System	bare computer

I, the undersigned, representing CAP Industry Limited, have implemented no deliberate extensions to the Ada Language Standard ANSI-MIL-STD-1815A in the compiler(s) listed in this declaration. I declare that CAP Industry Ltd is the owner of record of the Ada language compiler(s) listed above and, as such, is responsible for maintaining said compiler(s) in conformance to ANSI-MIL-STD-1815A. All certificates and registrations for Ada language compiler(s) listed in this declaration shall be made only in the owner's corporate name.

Kaye Ashworth — Date: 25-5-88

Name of Person signing
Title : Project Manager

*Ada is a registered trademark of the United States Government (Ada Joint Program Office).

Owner's Declaration

I, the undersigned, representing CAP Industry Ltd, take full responsibility for the implementation and maintenance of the Ada compiler(s) listed above, and agree to the public disclosure of the final Validation Summary Report. I further agree to continue to comply with the Ada trademark policy, as defined by the Ada Joint Program Office. I declare that all of the Ada language compilers listed, and their host/target performance, are in compliance with the Ada Language Standard ANSI/MIL-STD-1815A.

K Ashworth

Date: 16-6-88

Name of Person signing :

Title : Project Manager

Name of Base Compiler Owner : CAP Industry Ltd.

Notes on completion of Declaration of Conformance and Owner's Declaration.

Owner of record means the legal owner of the compiler.

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in in Chapter 13 of the MIL-STD-1815A, and to certain allowed restrictions on representation clauses. The implementation-dependent characteristics of CAPTACS-E286 V2.1 are described in the following sections which discuss topics one through eight as stated in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). Implementation-specific portions of the package STANDARD are also included in this appendix.

```
package STANDARD is

  type INTEGER is range -32768 .. 32767;
  type LONG_INTEGER is range -2147483648 .. 2147483647;

  type FLOAT is digits 6 range -3.37E38 .. 3.37E38;
  type LONG_FLOAT is digits 15 range -1.67E308 .. 1.67E308;

  type DURATION is delta 2**-14 range -86400.0 .. 86400.0;
   -- DURATION'small = 2**-14

end STANDARD;
```

CAP

F APPENDIX F

The Ada language definition allows for certain target dependences in a controlled manner. This appendix, called Appendix F as prescribed in the LRM, describes implementation-dependent characteristics of the CAPTACS-E286 system.

F.1 Implementation-Dependent Pragmas

The implementation-dependent pragma COMMENT is used for embedding a sequence of characters into the object code. The syntax is:

```
pragma COMMENT ( <string-literal> );
```

where: <string-literal> represents the characters to be embedded in the object code.

Pragma COMMENT may appear at any location within the source code of a compilation unit. Any number of comments may be entered into the object code using this method.

The implementation-dependent pragma INTERRUPT is used for function-mapped optimizations of interrupts as described in Section 12.11.1.5. The syntax is:

```
pragma INTERRUPT (FUNCTION_MAPPING);
```

F.2 Implementation-Dependent Attributes

There are no implementation-dependent attributes.

F.3 Package SYSTEM

The current specification of the package is provided below.

```
package SYSTEM is

   type SEG_OFFSET is new INTEGER;
   type SEG_SELECTOR is new INTEGER;
   type ADDRESS is private;
   type SUBPROGRAM_VALUE is private;
   type NAME is (CAPTACS_E286);
   SYSTEM_NAME : constant NAME := CAPTACS_E286;
   STORAGE_UNIT : constant := 8;
   MEMORY_SIZE : constant := 2**24;

   --System-Dependent Named Numbers:

   MIN_INT      : constant := -(2**31);
   MAX_INT      : constant := (2**31) - 1;
```

CAP

```
      MAX_DIGITS   : constant := 15;
      MAX_MANTISSA : constant := 31;
      FINE_DELTA   : constant := 1.0 / (2**( MAX_MANTISSA - 1 ));
      TICK         : constant := 1.0 / (2 ** 10);

      --Other System-Dependent Declarations:

      subtype PRIORITY is INTEGER range 0..15:

   private

   -- Types ADDRESS and SUBPROGRAM_VALUE are private

   end SYSTEM;
```

F.4 Representation Clauses

CAPTACS-E286 supports the following representation clauses:

1) Length clauses: for enumeration and derived integer types 'SIZE attribute (LRM 13.2(a))

2) Length clauses: for access types 'STORAGE_SIZE attribute (LRM 13.2(b))

3) Length clauses: for task types 'STORAGE_SIZE attribute (LRM 13.2(c))

4) Length clauses: for fixed point types 'SMALL attribute (LRM 13.2(d))

5) Record representation clauses (LRM 13.4)

6) Address clauses for objects and entries (LRM 13.5(a))

Note: CAPTACS-E286 has a restriction that allocated objects must have a minimum allocation size of 16 bits, and the minimum alignment for a record representation clause is 16 bits.

F.5 Implementation-Generated Names

There are no implementation-generated names denoting implementation-dependent components.

F.6 Address Clause Expression Interpretation

Expressions that appear in Address specifications are interpreted as the address of the first storage unit of the object.

CAP

F.7 Unchecked Conversion Restrictions

Unchecked conversions are allowed between types (or subtypes) T1 and T2 provided that:

1) they are not unconstrained record or array types.

F.8 Implementation-Dependent Characteristics of the I/O Packages

1. Instantiations of DIRECT_IO and SEQUENTIAL_IO are supported with the following exceptions:

   * unconstrained array types

   * unconstrained types with discriminants without defaults

2. There is no support for a file system. DIRECT_IO always raises a run-time exception. SEQUENTIAL_IO and TEXT_IO are supported only for sequential devices.

   Any attempt to create or open a named or temporary file (other than the specific named sequential devices) raises NAME_ERROR if the file name is greater than 12 characters long, and USE_ERROR otherwise.

3. In DIRECT_IO, the type COUNT is defined as follows:

```
type COUNT is range 0..2147483647;
```

4. In TEXT_IO, the type COUNT is defined as follows:

```
type COUNT is range 0..2147483645;
```

5. In TEXT_IO, the subtype FIELD is defined as follows:

```
subtype FIELD is INTEGER range 0..1000;
```

6. Package LOW_LEVEL_IO is defined for the following types:

   device_type:

```
type PORT_ADDRESS is new INTEGER;
```

   data_types:

```
type DATA_BYTE is range 0..255;
type DATA_WORD is range INTEGER'first..INTEGER'last;
type DATA_BYTE_ARRAY is array (NATURAL range <>)
                                          of DATA_BYTE;
type DATA_WORD_ARRAY is array (NATURAL range <>)
                                          of DATA_WORD;
```

CAP

F.9 Package MACHINE_CODE

Package MACHINE_CODE is not supported.

F.10 Language-Defined Pragmas

The language-defined pragmas CONTROLLED, ELABORATE, INTERFACE (ASSEMBLY_INTERFACE), LIST, PACK, PAGE, PRIORITY, SHARED and SUPPRESS are supported. The other language-defined pragmas, if included in Ada source, will have no effect.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name_and_Meaning	Definition
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	1..199=>'A',200=>'1'
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	1..199=>'A',200=>'2'
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	1..100=>'A',101=>'3', 102..200=>'A'
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	1..100=>'A',101=>'4',102..200=>'A'
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	1..197=>'0',198..200=>'298'
$BIG_REAL_LIT A universal real literal of value 690.0 with enough leading zeroes to be the size of the maximum line length.	1..194=>'0',195..200=>'69.0E1'

Name_and_Meaning	Definition
$BIG_STRING1 A string literal which when catenated with BIG_STRING2 yields the image of BIG_ID1.	1=>'2',2..101=>'A',102=>'"'
$BIG_STRING2 A string literal which when catenated to the end of BIG_STRING1 yields the image of BIG_ID1.	1=>'"',2..100=>'A',101..102=>'1"'
$BLANKS A sequence of blanks twenty characters less than the size of the maximum line length.	1..180=>' '
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2147483645
$FIELD_LAST A universal integer literal whose value is TEXT_IO.FIELD'LAST.	1000
$FILE_NAME_WITH_BAD_CHARS An external file name that either contains invalid characters or is too long.	X}]!@#$^&~Y
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character or is too long.	XYZ*
$GREATER_THAN_DURATION A universal real literal that lies between DURATION'BASE'LAST and DURATION'LAST or any value in the range of DURATION.	100_000.0
$GREATER_THAN_DURATION_BASE_LAST A universal real literal that is greater than DURATION'BASE'LAST.	10_000_000.0

Name_and_Meaning	Definition
$ILLEGAL_EXTERNAL_FILE_NAME1 An external file name which contains invalid characters.	BAD_CHARACTER*^/%
$ILLEGAL_EXTERNAL_FILE_NAME2 An external file name which is too long.	1..120=>'A'
$INTEGER_FIRST A universal integer literal whose value is INTEGER'FIRST.	-32768
$INTEGER_LAST A universal integer literal whose value is INTEGER'LAST.	32767
$INTEGER_LAST_PLUS_1 A universal integer literal whose value is INTEGER'LAST+1.	32768
$LESS_THAN_DURATION A universal real literal that lies between DURATION'BASE'FIRST and DURATION'FIRST or any value in the range of DURATION.	-100_000.0
$LESS_THAN_DURATION_BASE_FIRST A universal real literal that is less than DURATION'BASE'FIRST.	-10_000_000.0
$MAX_DIGITS Maximum digits supported for floating-point types.	15
$MAX_IN_LEN Maximum input line length permitted by the implementation.	200
$MAX_INT A universal integer literal whose value is SYSTEM.MAX_INT.	2147483647

Name_and_Meaning	Definition
$MAX_INT_PLUS_1 A universal integer literal whose value is SYSTEM.MAX_INT+1.	2147483648
$MAX_LEN_INT_BASED_LITERAL A universal integer based literal whose value is 2#11# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	1..2=>'2#',3..197=>'0', 198..200=>'11#'
$MAX_LEN_REAL_BASED_LITERAL A universal real based literal whose value is 16:F.E# with enough leading zeroes in the mantissa to be MAX_IN_LEN long.	1..3=>'16#',4..196=>'0', 198..200=>'F.E#'
$MAX_STRING_LITERAL A string literal of size MAX_IN_LEN, including the quote characters.	1=>'"',2..199=>'A',200=>'"'
$MIN_INT A universal integer literal whose value is SYSTEM.MIN_INT.	-2147483648
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT or LONG_INTEGER.	SHORT_SHORT_INTEGER
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFE#

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 27 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

B28003A:	A basic declaration (line 36) wrongly follows a later declaration.
E28005C:	This test requires that 'PRAGMA LIST (ON);' not appear in a listing that has been suspended by a previous "pragma LIST (OFF);"; the Ada Standard is not clear on this point, and the matter will be reviewed by the ALMP.
C34004A:	The expression in line 168 wrongly yields a value outside of the range of the target T, raising CONSTRAINT_ERROR.
C35502P:	The equality operators in lines 62 and 69 should be inequality operators.
A35902C:	Line 17's assignment of the nominal upper bound of a fixed point type to an object of that type raises CONSTRAINT_ERROR for that value lies outside of the actual range of the type.
C35904A:	The elaboration of the fixed-point subtype on line 28 wrongly raises CONSTRAINT_ERROR, because its upper bound exceeds that of the type.
C35904B:	The subtype declaration that is expected to raise CONSTRAINT_ERROR when its compatibility is checked against that of various types passed as actual generic parameters, may in fact raise NUMERIC_ERROR or CONSTRAINT_ERROR for reasons not anticipated by the test.
C35A03E:	This test assumes that attribute 'MANTISSA' returns 0 when applied to a fixed-point type with a null range, but the Ada Standard doesn't support this assumption.
C35A03R:	This test assumes that attribute 'MANTISSA' returns 0 when applied to a fixed-point type with a null range, but the Ada Standard doesn't support this assumption.

C37213H: The subtype declaration of SCONS in line 100 is wrongly expected to raise an exception when elaborated.

C37213J: The aggregate in line 451 wrongly raises CONSTRAINT_ERROR.

C37215C:
C37215E:
C37215G:
C37215H: Various discriminant constraints are wrongly expected to be incompatible with type CONS.

C38102C: The fixed-point conversion on line 3 wrongly raises CONSTRAINT_ERROR.

C41402A: 'STORAGE_SIZE' is wrongly applied to an object of an access type.

C45332A: The test expects that either an expression in line 52 will raise an exception or else MACHINE_OVERFLOWS is FALSE. However, an implementation may evaluate the expression correctly using a type with a wider range than the base type of the operands, and MACHINE_OVERFLOWS may still be TRUE.

C45614C: REPORT_IDENT_INT has an argument of the wrong type (LONG_INTEGER).

A74016C:
C85018B:
C87B04B:
CC1311B: A bound specified in a fixed-point subtype decC85018Blies outside that calculated for the base type, raising CONSTRAINT_ERROR. Errors of this sort occur re lines 37 and 59, 142 and 143, 16 and 48, 252 and 253 of the four tests respectively (and possibly elsewhere).

BC3105A: Lines 159..168 are wrongly expected to be incorrect; they are correct.

AD1A01A: The declaration of subtype INT3 raises CONSTRAINT_ERROR for implementations that select INT'SIZE to be 16 or greater.

CE2401H: The record aggregates in lines 105 and 117 contain the wrong values.

CE3208A: This test expects that an attempt to open the default output file (after it was closed) with mode IN_FILE raises NAME_ERROR or USE_ERROR; by Commentary AI-00048, MODE_ERROR should be raised.